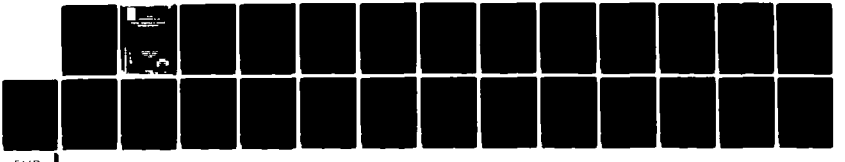
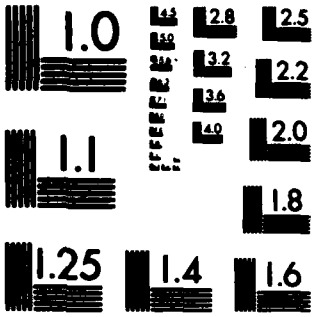
AD-A141 294 RESEARCH USING A DISCUS BASED NETWORK EMULATOR(U) ROYAL 1/1
SIGNALS AND RADAR ESTABLISHMENT MALVERN (ENGLAND)
T G CONNELLY ET AL. MAR 84 RSRE-MEMO-3539 DRIC-BR-91414

UNCLASSIFIED F/G 17/2 NL



END
DATE FILMED
7 84
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ROYAL SIGNALS AND RADAR ESTABLISHMENT

MEMO 3539

Title: RESEARCH USING A DISCUS BASED NETWORK EMULATOR

Authors: T G Connelly and M P Griffiths

Date: March 1984

SUMMARY

This memo describes a proposal to use the DISCUS multi-microprocessor system developed at RSRE as a tool for emulating a circuit switched network. The research which could be undertaken using such an emulator is outlined, and the advantages of this approach are described. An analysis is presented to show how the current system may be used with little or no modification.

CONTENTS

APPENDICES

1 INTRODUCTION

The main purpose of this paper is to propose a method of emulating a multi-node communications network using the DISCUS multi-processor system[1,2,3,4], one or more processors being used to represent each node. The use of such a network emulator would facilitate the study of the behaviour of networks containing a large number of nodes, and would provide a simpler and lower cost alternative to the construction of a "real" network of comparable size.

However, as a basis for a discussion of the emulator and its use, an attempt will first be made to define a programme of research into communications networks, to identify the experiments which need to be performed to achieve the aims of this programme, and to describe (briefly) the apparatus required to perform the experiments.

The Army has, up to the present time, used circuit switched networks for its communications. Therefore, for the purposes of this paper, the term "network" refers to a dynamic, circuit-switched communications network consisting of a number of (more or less identical) computer-controlled switching nodes, which are connected to a number of (more or less identical) subscriber terminals. However, there is no reason why the methods proposed here should not be used to emulate other types of network, (e.g. packet switched or message switched) but different performance parameters may have to be used.

2 NETWORK RESEARCH AIMS

2.1 Functions of the Network

(a) The main function of the network is to set up circuits over which the user's traffic can be passed, and to clear down such circuits when they are no longer required.

(b) In addition, because the network is dynamic, there is a subsidiary requirement to pass from one node to another information concerning the status of nodes, subscribers, and links, and the connectivity of the network. (That is, to maintain and update a network data-base for system management purposes.)

2.2 Overview of Research Objectives

The fundamental aim of the research is to discover how to construct and use a communications network so as to perform the above functions efficiently, reliably and cost effectively. The research objectives may be discussed in under two headings:

(1) Mechanisms for setting up and clearing down calls

(2) Methods of manipulating the data-base, including methods of dealing with errors in the data, and faults arising from them.

2.2.1 Mechanisms for Setting up and Clearing Down Calls

The mechanisms used to set up and clear down calls are already well known. The research in this area will therefore largely be confined to deciding on a set of measurable parameters which describe how "good" any communications network is (e.g. probability of a call succeeding at the first attempt under given conditions), and measuring these parameters. Some indication of what parameters should be used may be given by the results of the Traffic Loading and Routing Model (TLRM) experiments[5], but it should be noted that these are concerned only with a static network.

The results of this characterisation exercise could then be used as a baseline from which to measure the effects of, for example, network reconfiguration, changes in the data-base structure and the method of data base updating, and errors in the data-base.

2.2.2 Network Data-base Manipulation

Static networks require a data-base which associates subscriber numbers with the communications channels by which they access the network, and which describes the inter-node connectivity. This data is unchanging, and therefore does not need to be passed from one node to another.

Dynamic networks, however, present a much more complex problem, since subscribers and nodes are mobile, and the whole system can thus be reconfigured. This implies that a large amount of data describing the network has to be kept up to date as reconfiguration occurs, and the relevant updates distributed to a large number of nodes. In a tactical area military communications system, the quantity of such data which would have to be transferred between nodes in the event of one node taking over another's function could be very large. Such transfers may not be uncommon in a system which may have to reconfigure rapidly and frequently in response to the changing tactical situation.

2.2.2.1 Data-base Updating

It will be necessary to devise some means of measuring how far out of date a node's data-base is, and what effect its being out of date has on the grade of service provided to the user by the whole of the network. Evidently each node must have at least one method of determining when its data-base requires updating, of obtaining updating information from other nodes, and of supplying other nodes with updating information. Some form of protocol for the exchange of updating information has to be established. Tests are required to determine whether the chosen protocol is sufficient, especially under reconfiguration, fragmentation and link failure conditions.

2.2.2.2 The Effects of Errors

As in all systems, the network is subject to sources of error which, when an attempt is made to use the erroneous data, will give rise to user-observed faults. Because data is transferred from one node to another, such errors can easily propagate (and perhaps multiply), so that an error may be detected in a node separated by several links from that in which the error first occurred. Therefore, the design of the network must take this into account, and must minimise the effects of these errors on the user. This involves developing techniques which:

(a) Minimise the occurrence of errors in the first place.

(b) Detect an error as soon as possible after it occurred, (and preferably before it has had a chance to produce a fault)

(c) Isolate the error as far as possible.

(d) Attempt to recover the network to an error free state.

2.2.2.3 Data-base Optimisation

Initially the network to be studied will use a data-base consisting of a somewhat simplified version of the data-base used in IAS[6]. It is expected that possibilities for optimisation, both in terms of the quantity of data stored and the ease with which it can be accessed, will become apparent in the course of work on other aspects of network research. Modifications of the data-base could then be made, and their effect on network performance measured.

2.2.3 Construction of a Representation of a Network

Before any of the above work can be undertaken, some apparatus which can be used to represent a network will have to be devised. The various options for this are discussed in Section 4, in the light of details of the experiments which it is envisaged will be performed, and which are outlined in Section 3.

2.3 Summary of Research Objectives

From the preceeding discussion the main objectives of the network research can be summarised as follows:

(a) The definition of a set of measurable parameters which describe how "good" a communications network is, and the possible comparison of networks in terms of these parameters.

(b) The construction of a network, and a means of stimulating it with traffic and analysing its performance in terms of the chosen set of parameters.

(c) The development of a means of keeping the data-base up to date and consistent throughout the network under all conditions, particularly during reconfiguration or fragmentation.

(d) The optimisation of the network data-base, in terms of the quantity of data stored and the ease with which it can be accessed.

(e) The minimisation of the occurrence of errors, the development of means of detecting errors and isolating them, and of recovering the network to an error-free state.

3 EXPERIMENTAL PROGRAMME

Before any experimental work can proceed, it will be necessary to have available at least one representation of a network with which to work, and to define the parameters in terms of which the network is to be characterised. These parameters can be divided into two types; those which are applicable to virtually all communications networks, and those which apply only to dynamic networks. Examples of these two parameter types are listed below, but the list is by no means exhaustive.

(1) Mean time taken to set up a call, for a given traffic loading (i.e. for a given rate at which call attempts are made).

(2) Percentage of calls failing, for a given traffic loading.

(3) Number of links per route in excess of the shortest path.

(4) Signalling channel and traffic channel occupancy per link.

(5) Network partitionability. i.e. the extent to which service to the user can be maintained when the network has become fragmented into a number of smaller networks[8].

(6) Network capacity. This may be different to node capacity since, even if nodes are non-blocking, the network can still block because the amount of traffic it can support is limited by the number of links and the number of nodes[8].

(7) Amount of service and network complexity. As the number of users of a network is increased, the grade of service to those users tends to stay constant until a limit is reached, when the grade of service begins to drop. The grade of service may be restored to its original level by increasing the complexity of the network (number of links, number of nodes, etc) and therefore "amount of service" may be defined as some kind of product of grade of service and the number of users to which the service is provided. However, the greater the amount of service the more complex the network becomes and, particularly in the case of a dynamic network, the more difficult it becomes to manage. Indeed, it may be that as the network becomes more complex, the amount of management traffic increases to such an extent that no further gain in amount of service provided is achieved. Thus a measure of the complexity of a network would be a useful parameter, as would a limiting complexity, if such an effect exists.

(8) Equilibrium update queue length, (i.e. the total amount of data in the network which needs to be passed to other nodes but has not yet been fully distributed), for a given rate of generation of new data.

(9) Time taken to reduce a specified back-log of updating information to the equilibrium level, for a given new data generation rate.

(10) Maximum new data generation rate which the network can tolerate continuously without the amount of undistributed data growing indefinitely.

It is also necessary to appreciate that any experiments performed to investigate the effects of "errors" on network performance will involve both deliberately introduced errors and "accidental" errors, the exact nature of which is unknown to the experimenter. These "accidental" errors will, of course, also be present in experiments where no deliberate errors are introduced, (see Appendix A).

3.1 Characterising the network

Using whatever network representation(s) is/are adopted, a traffic load is applied to the network, and the general (static) parameters are measured for a fixed network connected in some simple configuration. (Initially perhaps just a 2-node network should be used, even though this implies that several of the previously listed parameters will be meaningless.) A large number of calls must be made if the results are to be statistically significant. The measurements should then be repeated for various levels of traffic loading and various (more complex) network configurations.

Updating information is then by some means injected into the data-base of one or more nodes, and the corresponding changes (if any) made to the network. The dynamic network parameters can then be measured. These measurements should be repeated for various different traffic loads, network configurations, and rates of generation of updating information. A large number of calls will again be required under each set of circumstances if the results are to be statistically significant. (In addition, it may be informative to measure the dynamic parameters in the complete absence of normal user traffic.)

It should be noted that the network characterised by the above measurements will contain an unknown number of "design errors". If and when these are eventually discovered and corrected, the characterisation measurements will have to be repeated for the "improved" network.

3.2 Data-base Updating

One of the initial aims of the experiments on data-base updating is to determine whether the chosen protocol used to disseminate system management information throughout the network is capable of keeping the data-base held in each node sufficiently up to date and consistent to avoid user-detectable faults in all circumstances. If this is to any significant extent not so, then further work on the updating protocol will be necessary.

If the rules governing data-base management are sufficiently comprehensive, we shall be in a position to estimate which circumstances are most likely to cause the Network DBM protocol to fail. (The most potentially troublesome situations are likely to arise out of network fragmentation, reconfiguration, and errors in the data-base.) Experiments will be required in which these "worst-case" situations or sequences of events can be set up dynamically and the exact response of the network noted by a "blow-by-blow" tracing facility. Re-measurement of certain of the fundamental network parameters under the estimated "worst-case" conditions would also be useful.

3.3 The Effects of Errors

As outlined in Appendix A, errors may be considered as being of three types: design errors, permanent errors, and transient errors. An error of any type may be accidental or deliberately introduced. In the following discussion we will not consider permanent errors, since these arise out of hardware breakdown, which can be minimised by good design practices, sound engineering and the use of redundant hardware where necessary.

In the experiments outlined in previous sections we have been concerned with measuring the performance of a network, including all its accidental design and transient errors, some of which (particularly design errors) may thereby be discovered. In the experiments described below we are more concerned with the effects of transient errors, deliberately introduced to simulate problems of data corruption.

It is proposed that transient errors be investigated under the following headings:

(a) Error avoidance

(b) Measurement of network performance degradation due to known errors

(c) Error detection

(d) Isolation of errors

(e) Recovery to an error-free state

3.3.1 Error Avoidance

This is probably more a matter of the use of defensive mechanisms within each network node to ensure that transient errors do not arise from design inadequacies, rather than of network phenomena as such.

3.3.2 Performance Degradation due to Deliberately Introduced Errors

These experiments consist of re-measuring the network characterisation parameters in the presence of known error patterns.

The main problem is to introduce errors which realistically represent those which would occur under field conditions, both in terms of error rate and pattern, and the places in the network at which the errors originate. The probable causes of transient errors in the data held by a node are not well known, but are likely to be very dependent on the actual hardware and software used, so that it may be difficult to establish likely sources of errors in one network by using any other network, "real" or emulated. However, the effects of errors which become included in a node's data-base, for whatever reason, can be measured. These errors can be conveniently introduced by deliberately corrupting data either on inter-node links or in a node's data-base. The errors can be introduced at a large number of different rates and in a large number of different patterns, and the degradation of network performance monitored.

3.3.3 Error Detection

This again is to some extent a matter of defensive mechanisms in network nodes rather than of network behaviour as such. Evidently such techniques as BCH coding and ARQ systems provide protection against data corruption on inter-node links, but further measures are needed within the data-base management software to minimise the effects of any errors which become included in the data-base, either through failure of the BCH and ARQ hardware, or because of some other problem within the node concerned. Such measures include range checking values retrieved from the data-base, and cross-checking data-base entries against each other where possible, which implies a certain amount of redundancy in the node's data-base.

However, if the data transmission checks and internal range checking etc. fail to detect an error, then a user-detectable fault may result. We then have a situation in which the user knows of a fault, but the system does not, so that it is important to provide some means for the user to communicate to his access node: (a) that there is a fault, and (b) possibly, in some circumstances, what the correct data should be.

Errors may also be detected by a node receiving two updates for the same data-base item which have the same edition number, but differ in content, due to some failure of the data-base updating protocol. In this case evidently at least one update will be in error.

The effectiveness of all these error detecting mechanisms can be determined by measuring what percentage of deliberately introduced errors are detected:

(a) before they give rise to user observable faults,

(b) by the user when a fault occurs, and

(c) not at all

Note that some errors may also be corrected in the normal course of system operation, without being detected.

3.3.4 Isolation of Errors

The objective is to discover methods of preventing errors from being propagated from one node to another. Error isolation is closely related to error detection, in that isolation requires that data identified as erroneous must not be passed on to other nodes, which implies that errors in the data-base should be detected as soon as possible after they occur.

To determine the effectiveness of the network in isolating errors, it will be necessary to measure how rapidly an error introduced at one point in the network propagates to other points under various conditions, and to discover how far (i.e. over how many inter-node links) an error propagates, on average, before being detected. Again, large numbers of experiments will be required to ensure statistical significance.

3.3.5 Recovery to an Error-free Situation

When a node detects an error in its data-base, by whatever means, several courses of action are available.

(a) Ignore the error, in which case it may be propagated to other nodes, and will eventually cause user observable faults.

(b) Isolate the error (i.e. not use or pass on the erroneous data), but make no attempt to recover to the "correct" state. This may degrade the service provided by the network, which constitutes a form of user-observable fault. Eventually, perhaps the erroneous data will be updated correctly in the normal course of operations.

(c) Isolate the error and attempt to recover by re-constructing the correct state from the rest of the (correct) switch data-base. This implies a measure of redundancy in the data-base.

(d) Attempt to recover by requesting an update of the erroneous information from a neighbouring node.

(e) In addition to attempting to restore the switch data-base to a correct state to try to determine what other nodes may also hold the erroneous data, and to inform them of this situation.

Experiments are required to determine the effects of each of these strategies on network performance, as measured by the characterising parameters. Alternative (e) alone represents an attempt at "network recovery", rather than merely "local recovery", and a possible technique for achieving this is outlined below.

In devising a network recovery strategy, it should be remembered that, because a node may not use all the information contained in its data-base, erroneous data may be passed from one node to another without the error being detected in intermediate nodes. Thus an error originating at one place in the network may be detected by a node in an entirely different place.

Bearing this in mind, let us suppose that each separately updateable item in the data-base is tagged, when amended in a node by action originating at that node, with the node's identity, as well as the current edition number for the item. When the data item is passed to another node, the receiving node adds its own identity to the tag. If this were done each time the item was transferred, a complete history of the path of the item through the network would always be associated with the item. (Connectivity changes would, of course, complicate the problem.) Thus if an error were detected in a data item in a particular node, all previous nodes on its path could be advised of the potential error. When a node on the path was found which contained the item without the error, (this condition possibly being defined by the item having a different value but the same, or a higher, edition number) re-transmission to the node in which the error was detected could occur along the original path.

A complete history for every updateable item in the network data-base would almost certainly imply unacceptably large overheads in both storage space and transmission time, so that some fraction of the history would have to be used. This might consist of the identities of just a few nodes through which the data item passed most recently, together with the original amending node's identity in case a correct version could not be found along the preserved path segment, so that re-transmission from source could be requested. (Re-transmission could then take place by any path which included the preserved segment of the original one.) The length of the preserved path segment would depend on how far most errors propagated from their point of insertion to their point of detection, (i.e. on the results of the error isolation experiments discussed in Section 3.3.4).

If a node on the preserved path segment had performed some processing using the erroneous data, without detecting the error, secondary errors may have been produced, which could also have propagated to other nodes. These would either be corrected when the correction of the primary error caused the corrected version of the data containing the secondary errors to be flooded into the network, or might be detected and corrected by the "back-tracking" method described above.

Considerable care will be required both in devising a workable version of such a recovery strategy, and in performing experiments to determine its efficiency. As with most such performance optimising methods, there is a possibility of its doing more harm than good under some circumstances.

4 CONSTRUCTION OF A TACTICAL AREA NETWORK

A tactical area network consists of a set of processing elements or nodes connected together via communication links of known channel capacity in a constantly varying pattern. Any system upon which research into this type of network is done should therefore have (ideally) the following characteristics:

(1) A sufficiently large number of nodes to give a network of representative complexity, so that the interactions over multiple links may be studied.

(2) Be able to be stressed by passing variable quantities of traffic through the network in a varying pattern to test for bottlenecks etc.

(3) Have the ability to easily alter network connectivity while it is running, to simulate the effects which actually occur in the field.

(4) Perform the tests in as close to real time as possible.

(5) Incorporate the characteristics of the communications channels in the simulation, again to monitor for any bottlenecks.

(6) Take into account the passage of data-base information between nodes as well as the simulated traffic.

Bearing these points in mind we will now present methods of providing such a network.

4.1 A Single Processor Simulation

In this method an existing (large!) computer is used to represent the entire network. Each node is simulated by a process (or processes) which is scheduled by the machine's operating system, and the links by the use of certain procedures accessing areas of store simulating the passage of data between nodes. This network would be driven by some form of traffic generator and the results logged for later analysis.

The chief advantages of this approach are: (a) it is cheap in terms of capital cost (assuming the existence of a suitable computer) (b) it is very easy to create a network of arbitrary complexity, in terms of both nodes and connectivity, and (c) modification of connectivity is straightforward while the simulator is running. Indeed, the TLRM uses this approach, although its programming falls short of the ideals in that modifications to the network are only possible between runs, not during a simulation, thereby eliminating the possibility of studying transient effects. Consequently, a more representative system would have to be written from scratch if this method were adopted, which would be a lengthy procedure. The other disadvantage of this approach is that real-time simulation is not possible with just one processor.

4.2 A Real Multi-computer Network

This would consist of real switches communicating via real hardware over real cables. This, in theory, provides the most realistic way of studying a network and as such must be regarded as the best standard which we could have for comparison with a fielded system.

However, this approach suffers from a number of very significant disadvantages in experimental terms:

(a) It is expensive to construct, both in terms of equipment and man-power, and consequently the number of nodes is severely limited.

(b) Dynamic network reconfiguration is very difficult to carry out and monitor in a controlled fashion as it will involve the physical movement of cables, plugs etc and the co-operation of large numbers of staff simultaneously.

(c) Setting up a sequence of stimuli and monitoring the effects without modifying the system give rise to problems almost as complex as the network itself.

4.3 A DISCUS Multi-Computer Emulation

This would consist of an array of DISCUS processors with each node being emulated by one (or more) processors running, as far as possible, the already existing software which has been written for the Intermediate Access Switch (IAS)[6]. Inter-node communication would be represented by channels in global store accessed by procedures which simulate as closely as possible the characteristics of the real links in the emulated system.

This has a number of advantages over the two previous methods:

(a) As the nodes are actually separate computers running asynchronously and in parallel, as in a "real" network, the simulation can take place in something very close to real-time.

(b) As the cost of a DISCUS processor is much less than a complete switch, more nodes could be provided thereby giving a more representative network.

(c) For the most part, existing software and hardware can be used, with relatively little modification. However, a lot more data-base management software will have to be written in order to carry out useful experiments.

(d) The use of "global" store, accessible by all processors in the DISCUS system means that the stimulation and monitoring of the emulated network would be comparatively straightforward, and could be done by a separate processor(s) with minimal disturbance to the emulated nodes.

(e) Since the emulation runs in real-time, it would probably be feasible to connect the emulated network to a "real" network, thereby providing a larger number of nodes to support network data-base management experiments.

The primary snag with the network emulator is demonstrating the validity of the emulation when compared with results obtained using a "real" network. The inter-working of the emulated and any compatible "real" networks may be helpful in this respect.

5 NETWORK EMULATOR IMPLEMENTATION

Having shown in Section 4 that there are considerable advantages to be gained from the use of a DISCUS-based network emulator, the way in which such an apparatus might be implemented is examined below.

In order to achieve economy of both effort and capital cost, it is advisable to use as much as possible of the existing DISCUS hardware and software, and IAS applications software. For cost reasons it is particularly important to restrict hardware modifications and extensions to a minimum.

5.1 Emulation of Switching Nodes

The existing IAS applications software is functionally divided into a number of modules, each of which runs on a separate DISCUS processor. In emulating a network of IAS-like switches, the problem is to map applications functions on to DISCUS processors in such a way that a reasonably large number of nodes can be represented, and that a minimum of software modification is required.

The number of nodes which can be emulated is limited by four main factors:

(a) The number of DISCUS processors which can exist in a DISCUS system because of hardware constraints. This is at present 15, but can easily be increased to 24.

(b) The number of processors required to emulate each node.

(c) The effective bandwidth of the global store access mechanism, which limits the rate at which information can be transferred from one emulated node to another, and hence the number of inter-node links which can be emulated *in real time*.

(d) The total quantity of global store which can be provided in a DISCUS system.

The effects of these limitations are discussed below.

5.1.1 Mapping the IAS Software onto DISCUS Processors

Given that the DISCUS hardware configuration makes it physically impossible to build a system containing more than 24 processors, the most obvious choice seems to be to use a single processor to represent a node, thus maximising the size of the emulated network. A means of producing a software package containing all IAS functions, scheduled to run on a single processor, is provided by the test harness option in the DISCUS system generator. This at present produces code which runs on the Intel MDS, but only a fairly simple modification would be required to make it produce code which would run on a DISCUS processor.

5.1.2 Number of Processors Required to Emulate each Node

The most serious drawback of using a single processor to emulate a node is that it is doubtful whether the whole of the applications software, plus the switch data-base and a DISCUS operating system will fit into the 64 K bytes of local store available to a DISCUS processor. A preliminary estimate of the store required indicates that the software will fill some 116 K bytes which is almost double the amount which can be accommodated by a single processor.

There are a number of options available to overcome this problem:

(a) Use overlays from global store or some other medium (complex and slow).

(b) Use some sort of memory banking technique in local store. (This would involve considerable hardware effort and cost.)

(c) Modify the applications software to reduce its size. (This would require extensive modifications to be made, which would be very time-consuming and error-prone.)

(d) Use more than one DISCUS processor to represent each node. This involves either:

(1) Using global store to communicate between processors in the same node, via both DISCUS channels and arrays in the switch data-base. This may distort the traffic flow between nodes, and would involve some modifications to the DISCUS operating system to distinguish between inter-node and intra-node channels,

OR

(2) Using some method of communicating between processors in the same node which does not involve global store. This may be too slow, and would involve modifications to the operating system for the same reason as in (1) above. It will also require more hardware than alternative (1), thus increasing cost, and may involve some hardware development.

Of the above alternatives, (d) (1) seems the most promising, provided that there is sufficient global store access bandwidth to allow real-time emulation of inter-node links as well as other inter-processor communication.

The number of processors per node is then increased to 2 or 3, depending upon the level of detail that we wish to emulate. If we were to dispense with loop signalling entirely and merely emulate at the trunk level, then a considerable proportion of the software could be removed, and 2 processors will almost certainly be adequate. However, in order to perform the more interesting DBM experiments, a lot of DBM software will have to be written, and this may well exceed the address space of two processors. Therefore, in order to keep the system as flexible as possible, it is probably best to assume that 3 processors will be required for each emulated node.

Of the 24 processors in the DISCUS-based emulator, three will probably be required for control, monitoring etc. (see Section 5.2), leaving 21 to be used for node emulation. If three processors per node are used, this implies a maximum of seven emulated nodes. Two processors per node implies a maximum of ten nodes. Assuming that three processors per node are used, space in local store will not be critical, and the distribution of functions among processors can be decided almost entirely on the basis of minimising inter-processor communication requirements.

5.1.3 Effective Bandwidth of Global Store Access

Since all the information passed from one emulated node to another must be first "put" into global store and then "got" out again, the rate at which global store can be accessed determines the maximum rate of information flow between nodes. This in turn determines the number of bi-directional links, of any given bandwidth which can be emulated in real-time.

The average rate at which global store can be accessed by a processor depends on the number of processors requiring access and their distribution among the possible positions they can occupy in the DISCUS system. An analysis of this situation is given in Appendix B. The results of this analysis indicate that, assuming bi-directional links working at 8 Kb/s in each direction, the DISCUS global bus will support the information flow required by the links of a fully connected 7 node network, whilst leaving a similar amount of bandwidth for use in intra-node communication and monitoring purposes. We expect this to be perfectly adequate.

5.1.4 Constraints imposed by the Quantity of Global Store Available

Up to seven 32 K byte blocks of global store can be provided by the DISCUS hardware, i.e. a total of 224 K bytes. The maximum length of a channel provided by the DISCUS operating system is 255 bytes. (Channels will be used to represent inter-node links). For a fully connected 7-node network, 21 bi-directional links will be required, which implies 42 channels, taking up a total of 42 x 255 = 10.7 K bytes of global store. Thus there will be 224 - 10.7 = 213.3 K bytes of global store to be used for shared non-transient data. It is expected that this will be more than adequate.

5.2 Emulator Control Software

The three main requirements for control software are a traffic generator, a network configuration controller/emulator supervisor, and a results monitor.

5.2.1 Traffic Generator

This could either be a loop traffic generator, in which all the compelled signalling for a call was emulated, or a trunk traffic generator which originates and clears down calls at trunk level without emulating compelled signalling on local loops. It could probably run as an additional function within each node, in order to minimise the global store access bandwidth used for purposes other than inter-node communication. Another program running on each node would provide a "dummy" loop signalling load so as to give as realistic an environment as possible.

Alternatively a traffic generator running on a separate DISCUS processor, and communicating with emulated nodes via global store, could be used, provided that its use of global store access bandwidth does not become excessive.

A further alternative is to use a separate traffic generator communicating with emulated nodes by some means not involving global store, but this will require additional hardware, thereby increasing costs, and may involve some hardware development. It is therefore not to be recommended, unless other alternatives prove to have serious disadvantages.

5.2.2 Configuration Controller/Emulator Supervisor

This would be a software package running on a separate DISCUS processor, which could dynamically change the emulated system connectivity by altering global store to simulate the establishment and destruction of inter-node links. (This requires the development of a new software package, but is essential to the proposed emulation scheme.) It would also control the volume and pattern of traffic produced by the traffic generator, and the introduction of transient errors both on inter-node links and in that part of each node's data-base which is held in global store.

It would be convenient to combine the emulator supervisor (i.e. the man-machine interface) and the configuration control functions, and attach the emulator console to this processor, so that system parameters could be altered "manually" as necessary, and subsets of the results could be selected for immediate display as well as being logged for later off-line analysis (see below).

Manual alteration of parameters might include the ability to instruct individual processors representing nodes to trace certain of their operations (e.g. all messages received, or just a particular type of message, accesses to specified records, expiry of timeouts, etc.). The tracing specification could be set up before the start of an emulation, in much the same way as with a logic analyser.

5.2.3 Results Monitor

This provides the means of obtaining results from the emulator. It could be provided by each node reporting on its operations, in accordance with the trace specification described above. Useful information would be, for example, a log of all calls attempted with each reported event time-indexed, reasons for call failures, state of inter-node links (i.e. which are working and which are not). Reporting to a central logging device, which could transfer reported data to backing store (e.g. a floppy disc) would be convenient, and could be achieved using an MDS. Reporting could take place either via global store, provided the use of access bandwidth does not become excessive, or via some other external method, (which would involve some hardware cost and effort). Some results could probably be obtained from the configuration controller.

5.3 Outline Specification

From the preceding discussions it is now possible to specify, in fairly general terms, the main hardware and software components of the proposed network emulator.

Three DISCUS processors per emulated node will be required. This will allow at least 6 real-time inter-node links per node to be provided. (At least 3 are essential, since 2 links per node implies that nodes can be connected only in either a line or a ring.) In this configuration, up to seven nodes could be emulated, leaving 3 processors to perform the emulator control and monitoring functions described in Section 5.2. There would be sufficient global store access bandwidth, over and above that used for the inter-node links, to enable the control and monitoring processors to communicate between themselves and with the emulated nodes, and, of course, to allow inter-processor communication within each node.

The IAS applications software can be used in the emulated nodes with only a few minor modifications, these mainly concerning device handlers, although fairly significant amount of additional software for DBM will be needed. In addition, substantial re-arrangement of the DISCUS operating system and system generator will be required. It will probably not be necessary to modify the DISCUS hardware.

6 CONCLUSIONS

This document has proposed the development of an emulated network based on an array of DISCUS processors which has significant advantages over a "real" network both in terms of the range of experiments which are possible and the ease with which they can be performed, and in terms of cost.

To reiterate the advantages of the emulator therefore:

(1) The investigation of complex networks is possible at greatly reduced effort and cost compared with the use of a "real" network. This may be particularly significant in the area of network recovery and the error-path retracing procedure outlined in section 3.3.5.

(2) The fact that the whole network is centralised in one place means that the problems of results monitoring and alteration of network parameters and configuration are greatly simplified.

(3) The possibility exists of connecting real and emulated networks together, which may provide a more realistic environment for the emulated nodes.

Given these advantages, we believe that the DISCUS emulated network would provide a useful and cost-effective base for communications network research.

7 REFERENCES

[1] H S Field-Richards,
"The DISCUS Hardware System",
RSRE Report 82010

[2] M P Griffiths,
"The DISCUS Operating System",
RSRE Report 82009

[3] H S Field-Richards,
"The DISCUS Hardware Descriptions and Appendices",
RSRE Memo 3483

[4] H S Field-Richards and M P Griffiths,
"DISCUS Multiprocessor Enhancements",
RSRE Memo 3591

[5] "Introduction to the Traffic Loading and Routing Model",
Project Ptarmigan Document ZLA/157/01, RSRE(C), Jan 1977 (Unclassified)

[6] T G Connelly and M P Griffiths,
"Intermediate Access Switch - A Multiprocessor Application",
RSRE Report 83004

[7] K Jackson and H R Simpson,
"MASCOT",
RRE Tech. Note 778

[8] G Alia and E Martinelli,
"An Assessment of Circuit Switching Network Functionalities",
Annual Conference AICA, Bologna, 1980

APPENDIX A

Classification of Errors

It seems convenient to divide the sources of error in any system into three types. These are described, in the context of communications networks, below.

(a) Transient Errors

These are defined to be those caused by the corruption of some data items either in transit from one node to another, or within a node. The occurrence of such errors may be minimised by good design and engineering practices. Recovery from them is possible, in that, in principle, the system can be restored to a correct state by "back-tracking" to a known correct point and trying again, and in the case of data corrupted in transit, re-transmitting the data concerned.

(b) Permanent Errors

These are errors caused by failure of a particular item of hardware. Their occurrence can be minimised by careful design and construction of the system, and by the use of redundant hardware, arranged so that a standby can automatically take over the function of an item detected as being defective. Apart from this, such errors can only be corrected by the manual replacement of the faulty unit.

(c) Design Errors

These are errors caused by mistakes in the implementation of the original system specification. In general, the system can only be restored to a correct state by returning it to the factory for re-design. (*Software bugs are included in this category.*) Such errors can be minimised by designing the system within a rigidly defined scheme which does not allow designers to make the more obvious mistakes. Software partitioning into conceptually separate co-operating programs (as in DISCUS and MASCOT[7]) is a good example. Another is the use of a well-defined protocol for inter-node communications.

It may be useful to treat design errors discovered in fielded equipment as transient errors, since re-design in the field is not easily possible. Most design errors which come to light at such a late stage will be the result of a relatively rare set of circumstances. (If this were not the case the system would, hopefully, not have passed user trials.) It is therefore possible that a re-try after the detection of such an error might be successful, as the state of the system may have changed between the two attempts, so that he unexpected set of circumstances no longer exists. Consequently it is worth initially treating all detected errors as transient, although it is important that such errors be logged in some way to facilitate later investigation. Error detection, re-try, and error-logging mechanisms should therefore be included in the original design.

APPENDIX B

Analysis of Global Bus Bandwidth

In hardware terms, DISCUS consists of a number of "crates", (at present a maximum of 8), each of which can (electrically) accommodate four DISCUS processors, though this number is reduced to three for reasons of physical space. A crate bus arbitrator polls "request lines" from all the processors in the crate, and the outputs of these arbitrators are polled by the global bus arbitrator. (A full description of this mechanism is given in the DISCUS Hardware Description[1,3]) In very simplified terms, this can be represented by an arrangement of rotating "stepping" switches, as shown in fig 1. The global switch steps round at a rate of one port every 1 us and dwells on each port on which there is an indication from the crate "switch" that a processor requires access for a further 0.5 us, during which time one access to global store is made, (i.e. a single byte is either read or written). The "crate" switch cycles round more rapidly, and dwells on a port on which a processor requires access for as long as the processor takes to make one access. This description of course ignores the anomalous behaviour of the arbitrator circuitry which must inevitably occur occasionally because of the asynchronous nature of the global access requests and the sampling of the request lines.

Evidently there is a large number of possible configurations of up to 24 processors, which give rise to various levels of utilisation of the available global store access bandwidth. In order to get some idea of the amount of data which could be passed through global store by a processor, let us consider the following "worst case" situation.

Suppose a DISCUS system consists of 24 processors connected so that each crate contains three processors, as shown in fig 1. We assume that all processors access global store sufficiently often that there is always at least one processor waiting on each global arbitrator port. We now consider the exchange of data between two processors.

The most efficient way to implement a FIFO queue between two processors (which would be used to emulate an inter-node link, for instance), is by means of a circular buffer containing a read pointer, write pointer, and data buffer, all located in global store. The number of global store accesses required to put a byte into the buffer, or get a byte from it, is four: one to read the read-pointer, one to read the write-pointer (to check if the buffer is already full, in the case of a write, or empty, in the case of a read), one to "put" or get a byte, and one to re-insert the incremented read or write pointer.

Since there are three processors in a crate, each one will only manage to access global store every third time the global arbitrator points to the crate. Consequently it will take $3 \times 4 = 12$ global bus arbitrator cycles to "put" or "get" each byte.

The time taken for the global bus arbitrator to step from one port to another is 1 us, and the dwell time on each port on which a processor requires access is 0.5 us, so that the cycle time for the arbitrator is:-

$$8(1 + 0.5) = 12 \text{ us}$$

Hence the time taken for a processor to "get" or "put" a byte is $12 \times 12 = 144$ us.

To complete the transfer of a byte from one processor to another, the receiving processor will have to do the inverse of the "putting" processor, taking another 144 us. However, although it cannot get a byte out of global store which has not yet been put in, overlap is possibly due to the fact that the "putting" processor may put a second byte on the same global bus arbitrator cycle on which the "getting" processor gets the first. Consequently the time required to transfer a byte from one processor to another is 144 us, implying an I/O bit rate for each processor of:

$$\frac{1}{144 \times 10^{-6}} \times 8 = 55.5 \text{ kb/s}$$

(This includes information flowing both into and out of a processor)

If each processor represented an emulated node, the number of links it could have may be calculated as follows. Assume an inter-node link is bi-directional and works at 8 Kb/s in each direction, so that a link emulated in real-time will require a bandwidth of 16 Kb/s. Therefore the maximum number of fully-loaded links which can be allocated to each node in this configuration is:

$$\text{Integer part of } \left(\frac{55.5}{16}\right) = 3 \text{ links}$$

if real-time emulation is to be maintained.

This implies that the maximum number of links in the entire system in this configuration is:

$$\frac{3 \times 24}{2} = 36$$

In fact this figure is slightly less than the number of links which the total global store access bandwidth in this configuration would support, due to the fact that each processor has sufficient access bandwidth to support 3 and a bit links. The difference between the access bandwidth required to support three links per processor and the total bandwidth which can be provided in this configuration could be used for emulator control and monitoring purposes. This "spare capacity" may be calculated as follows:

Total available access bandwidth

$$= \frac{1}{2(\text{No of accesses per byte put or got})(\text{time for 1 access})}$$

$$= \frac{1}{2 \times 4 \times 1.5 \times 10^{-6}}$$

$$= 83.3 \text{ kbytes/s} = 666.6 \text{ kb/s}$$

Access bandwidth required to support 36 links

$$= 36 \times 16 = 576 \text{ Kb/s}$$

Therefore spare capacity = 666.6 - 576 = 90.6 Kb/s

This is however, a somewhat unrealistic configuration for a network emulator, in that it is unlikely that one processor would be sufficient to emulate a node, and that no allowance has been made for processors to run the controlling and monitoring software.

Taking the discussion of Section 5.1.1 into account, a more realistic configuration would be to have seven emulated nodes, each of three processors, (i.e. one crate per node, for convenience), plus a crate of three processors for monitoring and controlling the emulator. For this configuration, the number of links per node, (assuming only one processor in each node communicates with other nodes via the emulated links), can be calculated as follows.

We assume that each node accesses global store sufficiently often that there is at least one processor waiting on each global bus arbitrator port at all times. We also assume that the control and monitoring processors also access global store to this extent. Thus all 8 global bus arbitrator ports are in use, as shown in fig 1.

Four global store accesses are required to "put" or "get" a byte, which implies four global bus arbitrator cycles to transfer each byte from one processor to another.

Therefore time taken to transfer one byte = (8 x 1.5) x 4 us = 48 us

$$\text{Therefore I/O bit rate per node} = \frac{8}{48 \times 10^{-6}} = 167 \text{ Kb/s}$$

This will support:

Integer part of (167/16) = 10 bi-directional links per node

This is more than is required for a fully connected 7-node network (which would require 6 links/node).

Thus, if we allocate say, 6 links/node, we require a global store access bandwidth of: 6 x 16000 = 96 Kb/s for each node

This leaves 167 - 96 = 71 Kb/s of global store access bandwidth in each node for inter-processor communication within a node, and for communication between a node and the controlling and monitoring processors.

The total global store access bandwidth used to emulate links in a fully connected 7 node network is:

(No of bi-directional links) x 16 Kb/s = (7 x 6 x 16)/2 = 336 Kb/s

Since the total available bandwidth was calculated as 666.6 Kb/s, some 330 Kb/s, (i.e. about half the total), remain to be used for other purposes.

APPENDIX C

Glossary of Abbreviations

ARQ	Automatic Request to repeat
BCH	Bose-Chaudhuri-Hocquenghem
DBM	Data-Base Management
DISCUS	DIStributed Control uprocessor System
FIFO	First In, First Out
IAS	Intermediate Access Switch
MASCOT	Modular Approach to Software Construction, Operation and Test
MDS	Microcomputer Development System
TLRM	Traffic Loading and Routing Model

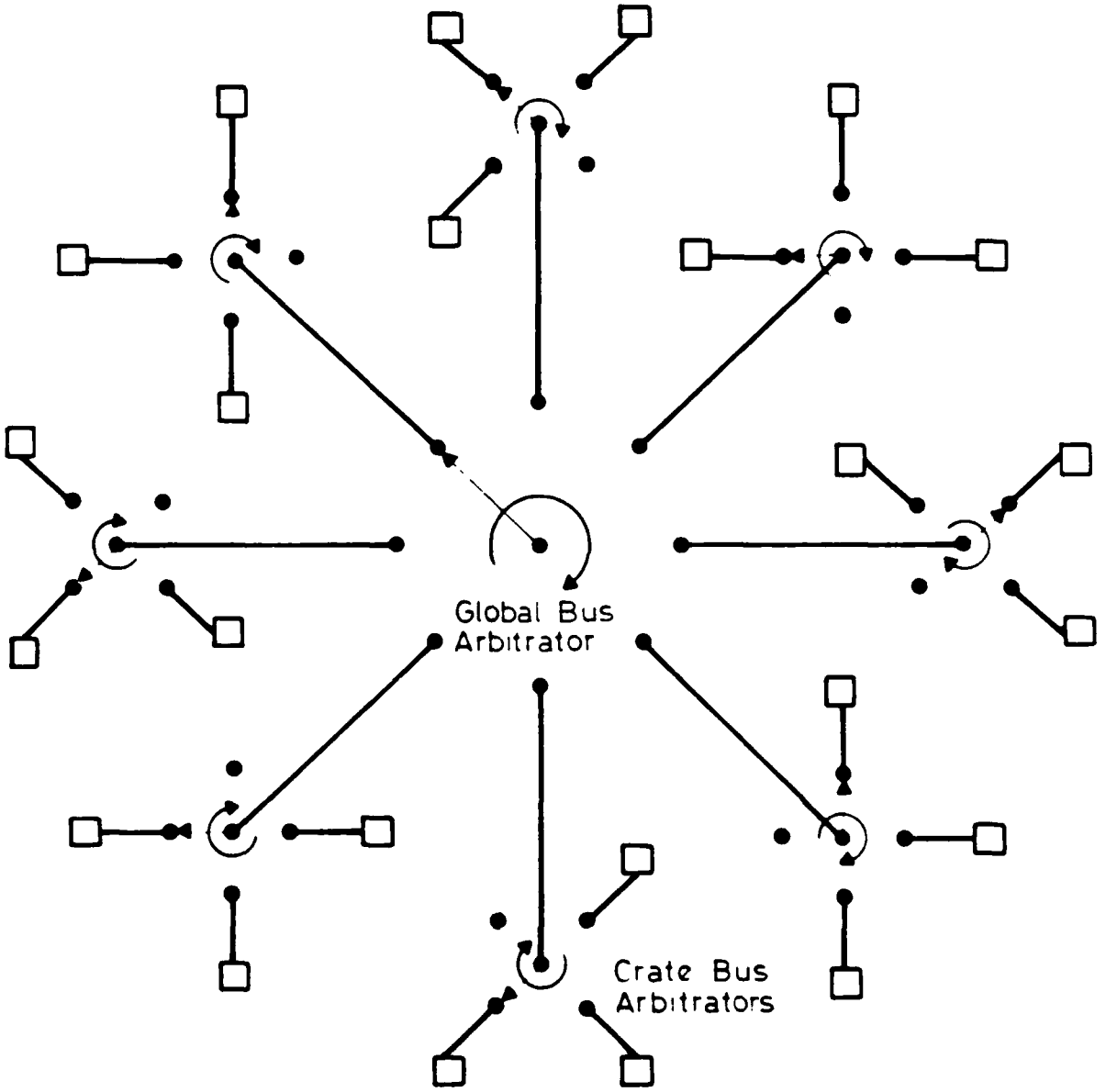


FIG 1 SIMPLE REPRESENTATION OF CRATE BUS AND GLOBAL BUS ARBITRATORS